Final Project Report

User Engagement

ISM 6362 – Big Data and Cloud-Based Tools

Soyeong Cha

Introduction

For this final project, I set out to design and implement a data analytics pipeline focused on understanding user engagement and feature adoption. While the technical challenge centered on using tools such as AWS S3, Databricks, PySpark, and Power BI, the real purpose of the work was to answer a key business question: how are users interacting with the platform, and are they adopting its features?

The dataset represented music streaming sessions, where customers listened to tracks by various artists. By processing this data, the goal was to measure not only how often customers returned and how long their sessions lasted, but also how they engaged with features that could indicate deeper adoption, such as exploring different artists or repeatedly using certain platform functions.

This focus on engagement and adoption mirrors real-world analytics challenges in technology and streaming businesses. Companies often release features without knowing whether users actually adopt them, and data pipelines like this are critical in providing the insights needed to make product decisions. While technical challenges shaped the exact implementation, the overarching aim was always to surface metrics that reflected user engagement and feature adoption in a meaningful, business-ready format.

Project Objectives

The main objective of this project was to design a pipeline that could track user engagement and feature adoption in a realistic streaming environment. Engagement was defined not only by how many sessions occurred and how many unique customers participated, but also by how often users returned and whether they interacted with different features of the system. By focusing on engagement and adoption, the project aligned technical outcomes with real business questions that product managers and business leaders would care about.

The first objective was data ingestion. Using Databricks Auto Loader, I needed to bring in raw session and event data from AWS S3 in a continuous and reliable way. This step was critical because the foundation of any engagement analysis is the activity logs that capture when and how users interacted with the platform. Without clean and timely ingestion, no higher-level insights about adoption could be produced.

The second objective was streaming transformation. Here, I normalized event timestamps, applied watermarking, and built five-minute session windows to capture active sessions and unique customers in near real time. This type of transformation allowed me to monitor engagement continuously, which is especially important for platforms where usage spikes can happen at any time. For example, understanding session windows could highlight patterns such as whether a feature launch led to immediate adoption or whether users slowly engaged with it over time.

A third objective centered on feature adoption metrics. Ideally, the pipeline would capture behaviors tied directly to adoption, such as users exploring new tracks, returning to listen to the same artist, or interacting with new functions offered by the platform. Due to challenges with calculation logic and file access, I was not able to fully implement this component, but the architecture was designed with this extension in mind. The ability to measure adoption is what transforms a simple session analysis into a deeper understanding of user behavior and product success.

Another key objective was data storage. I used Delta Lake “gold” tables as the target for curated results, ensuring that processed data was not only accurate but also available for consistent querying. This layer mattered because adoption trends need to be compared across time periods, cohorts, and features, all of which depend on reliable and query-friendly data storage.

Finally, the project included a visualization objective. The curated data was connected to Power BI to create interactive dashboards that displayed engagement and adoption metrics in a way that non-technical stakeholders could use. The ability to show sessions per window, active customers, and emerging adoption trends was important for translating technical work into business insights. Visualization closed the loop of the project by ensuring that the metrics could directly inform decision-making.

Alongside these technical goals, I also emphasized documentation and communication. A clear architecture diagram and this report were part of the project deliverables, ensuring that the overall design, the challenges encountered, and the insights gained could be understood by others. This reflected real-world expectations, where engineers and analysts not only build pipelines but also explain them in a way that supports collaboration across teams.

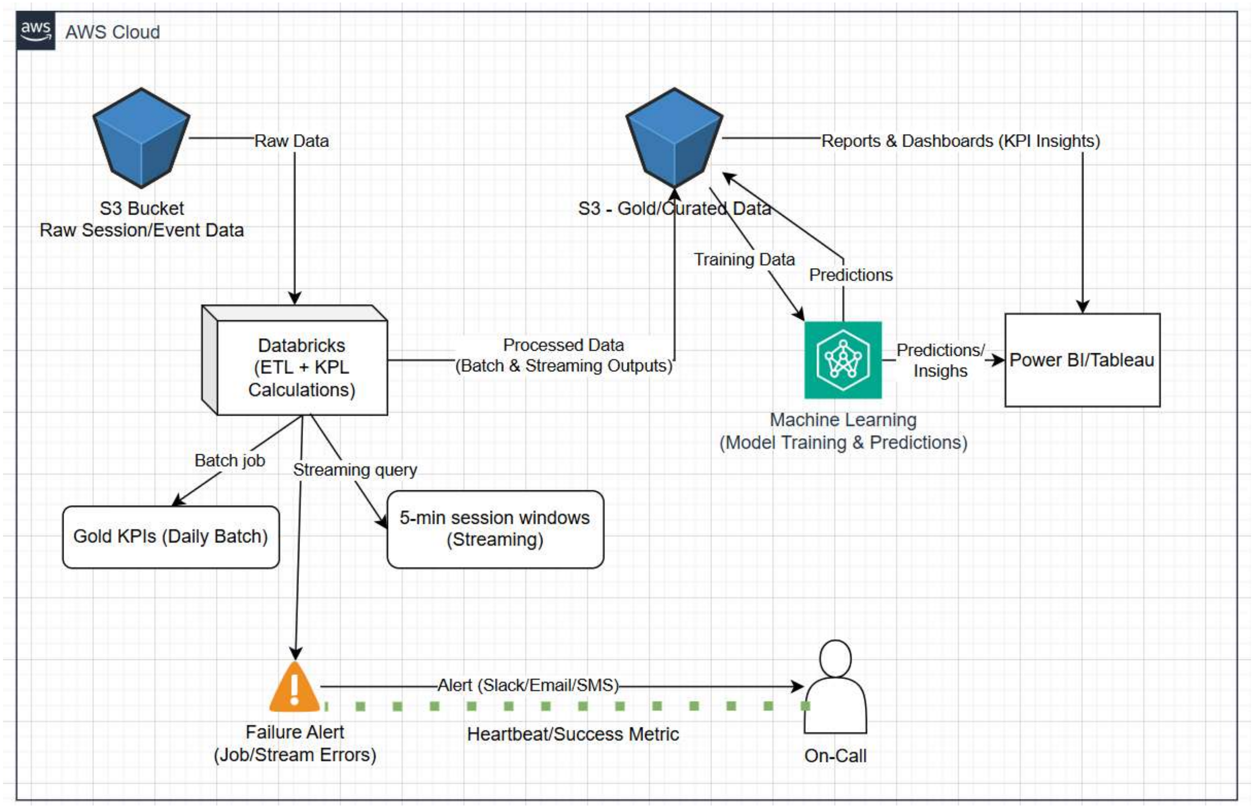


Figure 1. *End-to-End Data Pipeline Architecture.*

Data Sources and Structure

All project data was stored in parquet format within the AWS S3 bucket. Parquet was chosen because it is a columnar storage format optimized for analytical queries, making it efficient for large-scale data processing with PySpark. The datasets included:

- sessions.parquet – containing user session information (session IDs, start times, customer references).
- customers.parquet – list of customers with identifiers and attributes.
- artists.parquet – details of artists such as IDs and names.
- tracks.parquet – metadata about tracks including track IDs and artist references.
- dates.parquet – supporting calendar and timestamp reference information.

- events.parquet – a larger, more complex parquet file representing raw event-level data (clicks, plays, interactions). This file was particularly challenging to process during streaming because of its size and schema complexity.

By using parquet for all sources, the project mirrored a realistic big data environment, where high-volume, append-only event logs are stored in S3 and consumed downstream by structured streaming jobs.

Technical Implementation

1. Setting Up the Environment

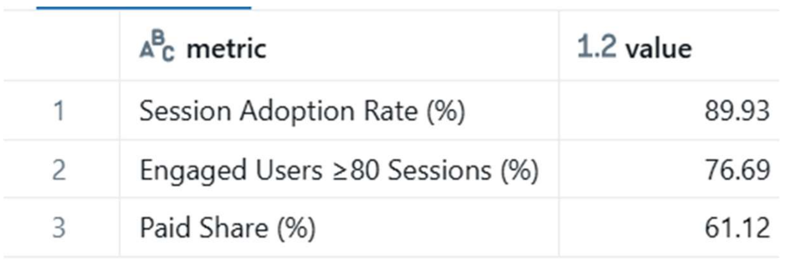
The project was implemented using Databricks notebooks as the central environment for development and execution. Databricks provided the flexibility to work with PySpark while maintaining an integrated workspace for both code and results. Data storage was managed through an AWS S3 bucket, where the session and event files were uploaded. To support structured streaming, a checkpoint directory and schema location were configured. These locations were essential for ensuring that the pipeline could handle incremental processing, maintain consistency, and recover gracefully in the event of errors. This environment setup represented the foundation for the entire pipeline.

2. Streaming Pipeline (Step 1 & 2)

The pipeline began with Databricks Auto Loader, which allowed parquet files in the S3 bucket to be ingested automatically, including both new and existing files. Auto Loader was particularly useful for its schema inference capabilities and its ability to handle evolving data structures without requiring extensive manual intervention. Once ingested, the session data was normalized by converting string-based timestamps into proper event-time columns. This step ensured that downstream transformations relied on consistent temporal data.

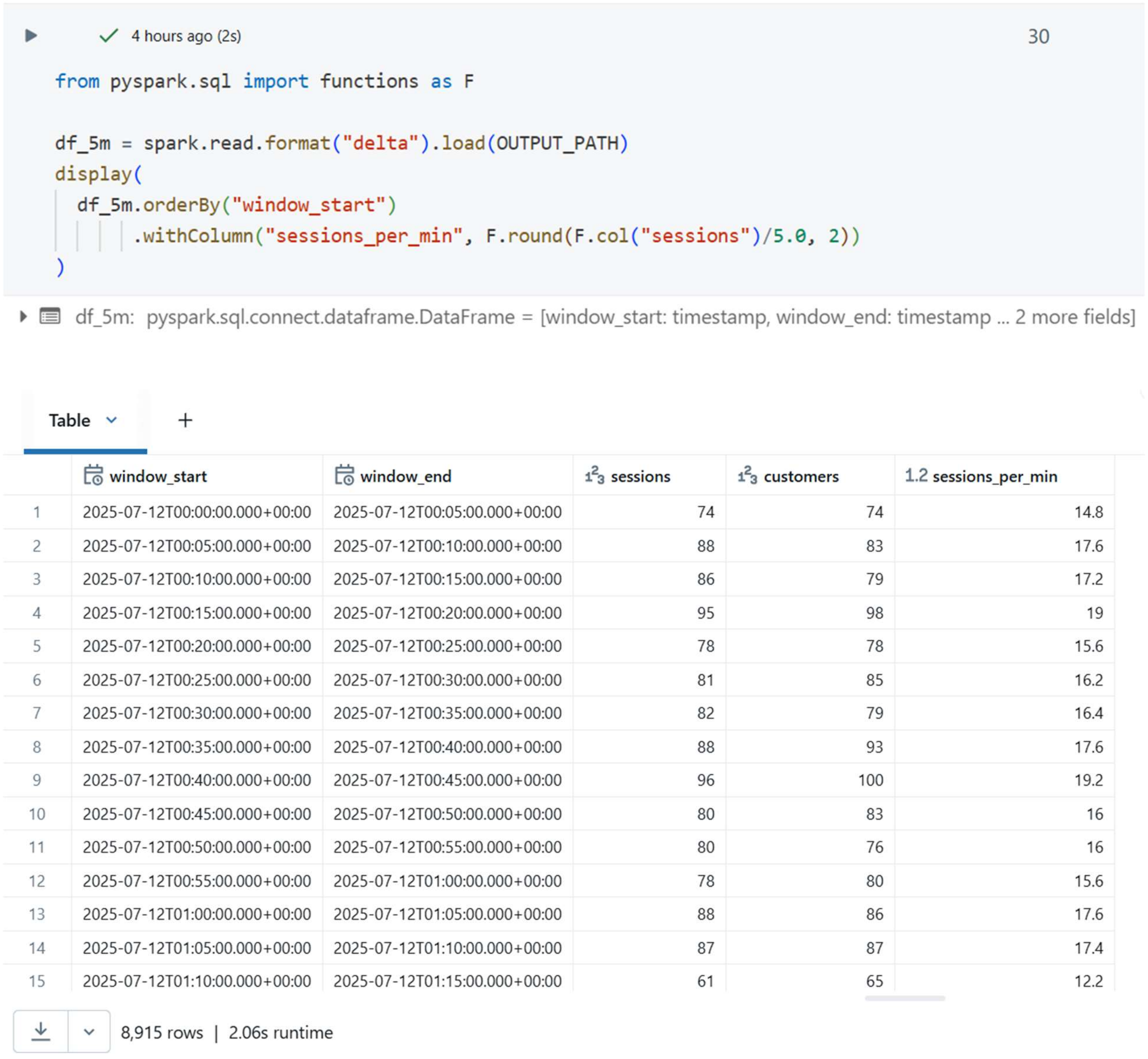
After normalization, I applied a five-minute tumbling window to group events into fixed intervals. A one-hour watermark was introduced to manage late-arriving data, ensuring that results were accurate without risk of double counting. Within each five-minute window, I calculated two primary metrics: the total number of sessions and the approximate number of distinct customers. These metrics provided insight into both overall activity and unique engagement over time.

```
stream = (spark.readStream
.format("cloudFiles")
.option("cloudFiles.format", "parquet")
.option("cloudFiles.includeExistingFiles", "true")
.option("cloudFiles.schemaLocation", SCHEMA_PATH)
.load(INPUT_PATH))
```



	metric	value
1	Session Adoption Rate (%)	89.93
2	Engaged Users ≥80 Sessions (%)	76.69
3	Paid Share (%)	61.12

Figure 2. *KPIs*



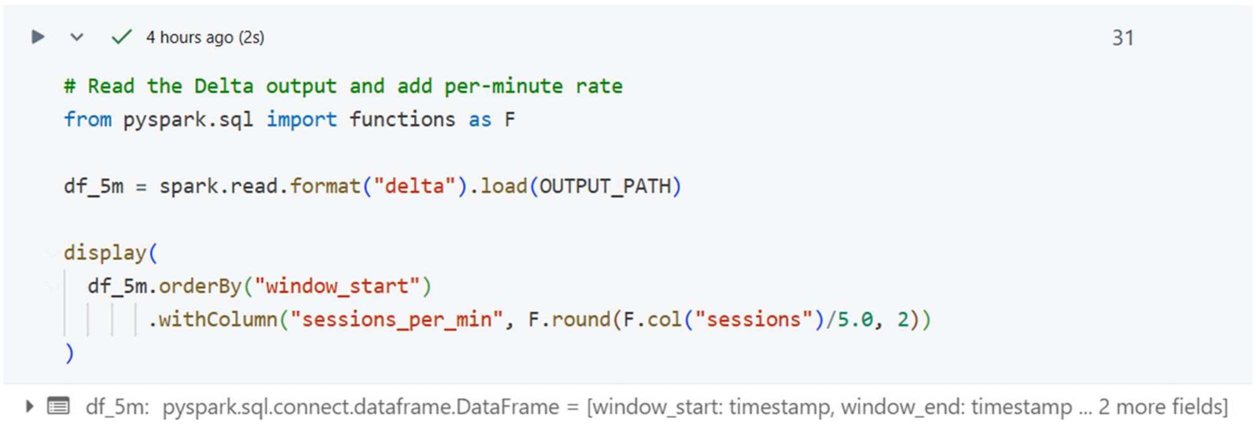
```python
from pyspark.sql import functions as F

df_5m = spark.read.format("delta").load(OUTPUT_PATH)
display(
  df_5m.orderBy("window_start")
       .withColumn("sessions_per_min", F.round(F.col("sessions")/5.0, 2))
)
```

```
df_5m:  pyspark.sql.connect.dataframe.DataFrame = [window_start: timestamp, window_end: timestamp ... 2 more fields]
```

	window_start	window_end	sessions	customers	sessions_per_min
1	2025-07-12T00:00:00.000+00:00	2025-07-12T00:05:00.000+00:00	74	74	14.8
2	2025-07-12T00:05:00.000+00:00	2025-07-12T00:10:00.000+00:00	88	83	17.6
3	2025-07-12T00:10:00.000+00:00	2025-07-12T00:15:00.000+00:00	86	79	17.2
4	2025-07-12T00:15:00.000+00:00	2025-07-12T00:20:00.000+00:00	95	98	19
5	2025-07-12T00:20:00.000+00:00	2025-07-12T00:25:00.000+00:00	78	78	15.6
6	2025-07-12T00:25:00.000+00:00	2025-07-12T00:30:00.000+00:00	81	85	16.2
7	2025-07-12T00:30:00.000+00:00	2025-07-12T00:35:00.000+00:00	82	79	16.4
8	2025-07-12T00:35:00.000+00:00	2025-07-12T00:40:00.000+00:00	88	93	17.6
9	2025-07-12T00:40:00.000+00:00	2025-07-12T00:45:00.000+00:00	96	100	19.2
10	2025-07-12T00:45:00.000+00:00	2025-07-12T00:50:00.000+00:00	80	83	16
11	2025-07-12T00:50:00.000+00:00	2025-07-12T00:55:00.000+00:00	80	76	16
12	2025-07-12T00:55:00.000+00:00	2025-07-12T01:00:00.000+00:00	78	80	15.6
13	2025-07-12T01:00:00.000+00:00	2025-07-12T01:05:00.000+00:00	88	86	17.6
14	2025-07-12T01:05:00.000+00:00	2025-07-12T01:10:00.000+00:00	87	87	17.4
15	2025-07-12T01:10:00.000+00:00	2025-07-12T01:15:00.000+00:00	61	65	12.2

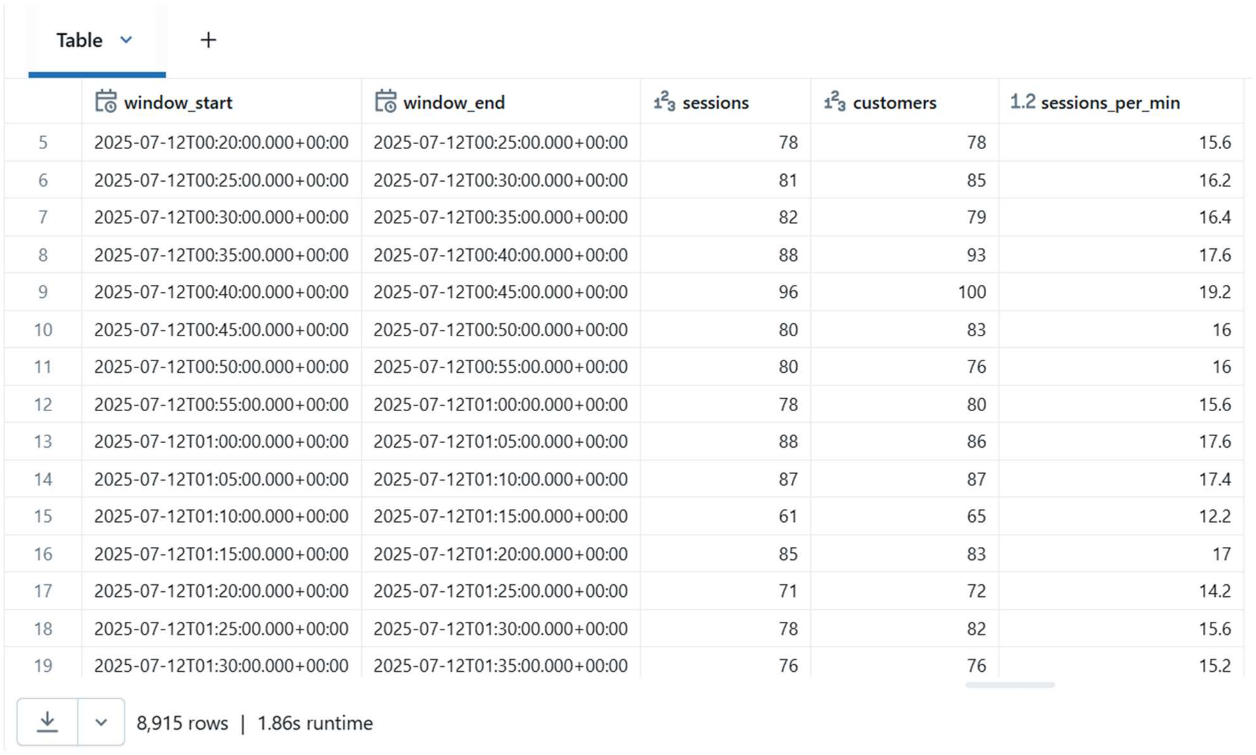
8,915 rows | 2.06s runtime

Figure 3. *5-Minute Session Trends.*



```python
# Read the Delta output and add per-minute rate
from pyspark.sql import functions as F

df_5m = spark.read.format("delta").load(OUTPUT_PATH)

display(
  df_5m.orderBy("window_start")
       .withColumn("sessions_per_min", F.round(F.col("sessions")/5.0, 2))
)
```

```
df_5m:  pyspark.sql.connect.dataframe.DataFrame = [window_start: timestamp, window_end: timestamp ... 2 more fields]
```

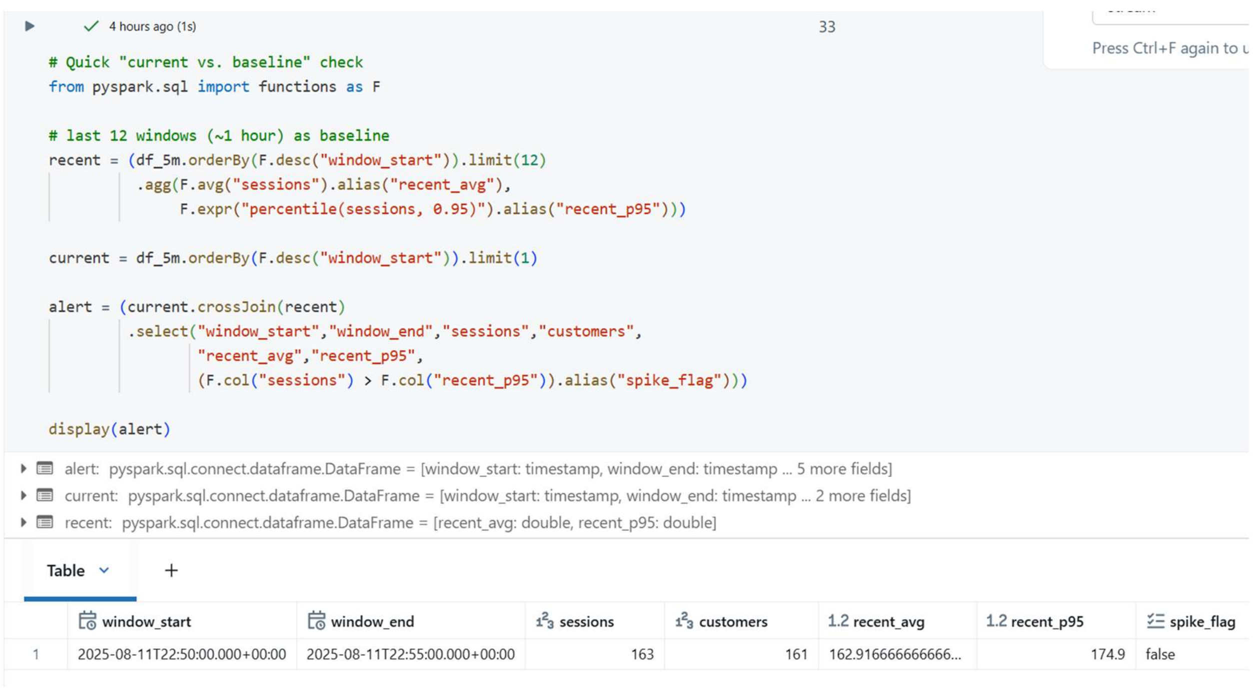


Table

	window_start	window_end	sessions	customers	sessions_per_min
5	2025-07-12T00:20:00.000+00:00	2025-07-12T00:25:00.000+00:00	78	78	15.6
6	2025-07-12T00:25:00.000+00:00	2025-07-12T00:30:00.000+00:00	81	85	16.2
7	2025-07-12T00:30:00.000+00:00	2025-07-12T00:35:00.000+00:00	82	79	16.4
8	2025-07-12T00:35:00.000+00:00	2025-07-12T00:40:00.000+00:00	88	93	17.6
9	2025-07-12T00:40:00.000+00:00	2025-07-12T00:45:00.000+00:00	96	100	19.2
10	2025-07-12T00:45:00.000+00:00	2025-07-12T00:50:00.000+00:00	80	83	16
11	2025-07-12T00:50:00.000+00:00	2025-07-12T00:55:00.000+00:00	80	76	16
12	2025-07-12T00:55:00.000+00:00	2025-07-12T01:00:00.000+00:00	78	80	15.6
13	2025-07-12T01:00:00.000+00:00	2025-07-12T01:05:00.000+00:00	88	86	17.6
14	2025-07-12T01:05:00.000+00:00	2025-07-12T01:10:00.000+00:00	87	87	17.4
15	2025-07-12T01:10:00.000+00:00	2025-07-12T01:15:00.000+00:00	61	65	12.2
16	2025-07-12T01:15:00.000+00:00	2025-07-12T01:20:00.000+00:00	85	83	17
17	2025-07-12T01:20:00.000+00:00	2025-07-12T01:25:00.000+00:00	71	72	14.2
18	2025-07-12T01:25:00.000+00:00	2025-07-12T01:30:00.000+00:00	78	82	15.6
19	2025-07-12T01:30:00.000+00:00	2025-07-12T01:35:00.000+00:00	76	76	15.2

8,915 rows | 1.86s runtime

Figure 4. *Streaming Session Activity (5-Minute Intervals).*



```
# Quick "current vs. baseline" check
from pyspark.sql import functions as F

# last 12 windows (~1 hour) as baseline
recent = (df_5m.orderBy(F.desc("window_start")).limit(12)
          .agg(F.avg("sessions").alias("recent_avg"),
               F.expr("percentile(sessions, 0.95)").alias("recent_p95")))

current = df_5m.orderBy(F.desc("window_start")).limit(1)

alert = (current.crossJoin(recent)
         .select("window_start","window_end","sessions","customers",
                 "recent_avg","recent_p95",
                 (F.col("sessions") > F.col("recent_p95")).alias("spike_flag")))

display(alert)
```

```
alert:  pyspark.sql.connect.dataframe.DataFrame = [window_start: timestamp, window_end: timestamp ... 5 more fields]
current:  pyspark.sql.connect.dataframe.DataFrame = [window_start: timestamp, window_end: timestamp ... 2 more fields]
recent:  pyspark.sql.connect.dataframe.DataFrame = [recent_avg: double, recent_p95: double]
```

Table

	window_start	window_end	sessions	customers	recent_avg	recent_p95	spike_flag
1	2025-08-11T22:50:00.000+00:00	2025-08-11T22:55:00.000+00:00	163	161	162.916666666666...	174.9	false

Figure 5. *Streaming Session Spike Alert.*

3. Safe Writing with Delta

To ensure reliability, the aggregated results were written into Delta Lake tables. However, writing data in a streaming context requires caution. If empty microbatches are written, downstream tables can be cluttered with unnecessary rows, which complicates analysis. To prevent this, I implemented a safeguard with the foreachBatch writer. The logic checked whether each batch contained rows before committing the write. Only non-empty batches were appended to the Delta table. This approach maintained both efficiency and data integrity while preventing the storage layer from being polluted with redundant data.

4. Machine Learning Demo

To round out the project, I added a simple machine learning model that predicts whether a customer is likely to be engaged. I kept the definition consistent with KPI #2: customers with at least 80 sessions in the July–August 2025 window were labeled as engaged, while everyone else was not. The idea wasn't to build a complex model, but to show how the data pipeline could be used for predictions, not just reports.

The setup was straightforward. I used the number of sessions each customer had in the analysis window as the single input feature. With that, I trained a logistic regression model in Spark ML, splitting the data so that most of it trained the model and the rest was used to check how well it worked. Even with just one feature, the model did a decent job, giving a solid AUC score and producing probabilities that indicated how likely each customer was to stay engaged.

What I like about this exercise is that it shows how even a small step into machine learning can create useful business insights. Instead of only describing what already happened, the system can point toward what might happen next. Knowing which customers are most likely to keep using the platform opens the door to more thoughtful decisions, like focusing retention efforts on people at risk of dropping off or rewarding those who are consistently active. It’s a simple demo, but it captures how predictive modeling can complement engagement metrics in a meaningful way.

5. Visualization

For the visualization component of the project, I used Power BI to connect directly to Databricks and build an interactive dashboard. The goal of the dashboard was to provide clear insights into user engagement and feature adoption, translating the processed data into metrics that a business or product team could easily interpret.

The dashboard displays several key measures. First, overall adoption was captured through the session adoption rate percentage, which showed how consistently users were adopting the platform over time. Another important metric was the paid share percentage, which indicated what proportion of users were engaging with premium or paid features. The engaged users percentage offered a complementary view, focusing on the depth of usage rather than just raw counts.

To help stakeholders see trends over time, I created line charts of sessions and customers by day, which revealed the rhythm of daily activity and highlighted anomalies such as sudden dips or spikes in usage. To complement this, bar charts broke down the number of sessions by month, providing an at-a-glance comparison of overall engagement levels in July versus August.

The dashboard also included interactive slicers for time windows, enabling users to dynamically adjust the analysis period. This feature allowed for flexible exploration of adoption trends, helping business users drill into specific periods such as before and after a feature launch.

Overall, the Power BI dashboard was an important part of the project because it closed the loop between technical data engineering work and business insights. While the pipeline was responsible for processing and curating data, the dashboard was the point where adoption and engagement trends became visible and actionable. It reinforced the idea that visualization is only as strong as the upstream pipeline: without clean data and clear naming conventions, these dashboards would not have been possible.

Results

The Power BI dashboard provided several insights into user engagement and feature adoption. The session adoption rate was consistently high, averaging around 89%, which suggests that most users were actively engaging with the platform once they had access. This is a strong indicator that features were being adopted and that the system was relevant to users' needs.

The paid share percentage, which averaged around 61%, highlighted that a significant portion of users were interacting with premium features. This finding is important for product and business stakeholders because it points to healthy conversion and monetization potential. However, it also suggests room for improvement in encouraging more users to transition into paid features, which could become a focus for future growth.

The engaged users percentage, at approximately 77%, demonstrated that most users were not just adopting features but were also actively returning and interacting with them. This indicates sustained usage rather than one-time or shallow adoption.

When broken down over time, the sessions by day graph showed stable engagement across the month with a noticeable drop mid-cycle, likely due to either a data gap or a usage anomaly. Identifying such dips is valuable for operational monitoring, as it can point to underlying system issues or periods of lower user activity that warrant further analysis.

The sessions by month comparison revealed that July had a higher session volume than August, suggesting either seasonal trends, variations in user behavior, or the impact of external factors. This highlights the value of continuous monitoring and visualization: trends are only visible when data is consistently processed and surfaced.

Together, these results confirmed that the pipeline was able to capture and present the core engagement signals needed to evaluate feature adoption. While further refinements could improve granularity (such as per-artist or per-feature breakdowns), the current analysis already provides a solid foundation for business decision-making.

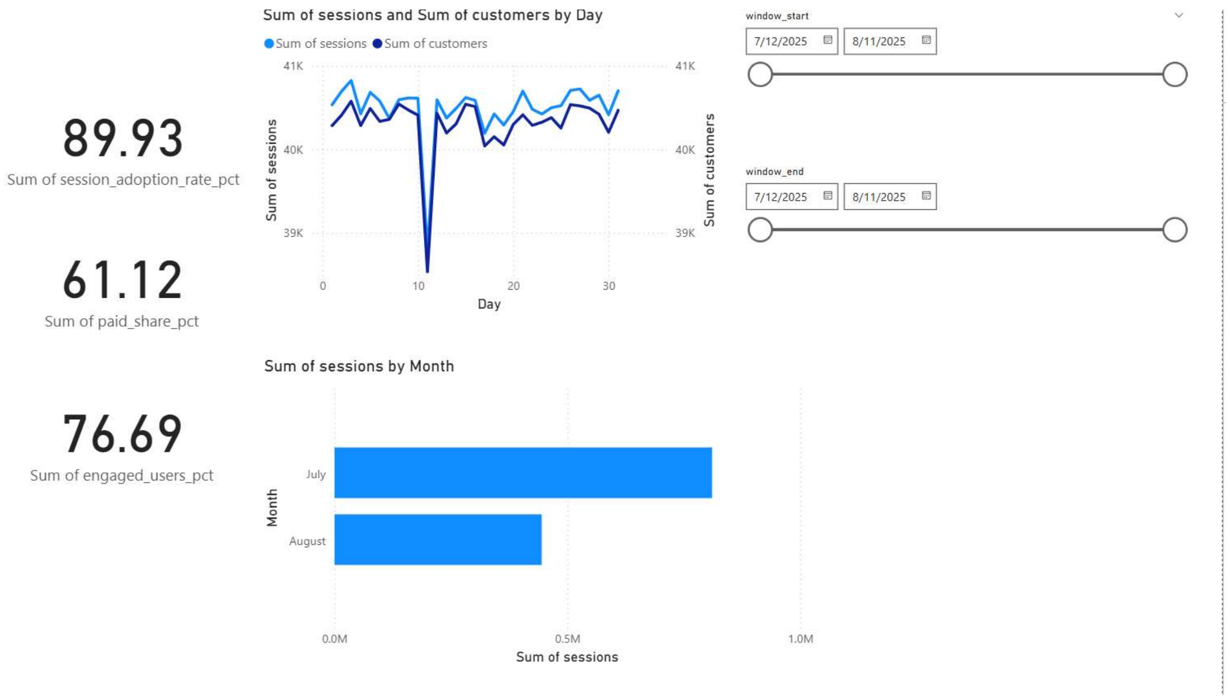


Figure 6. *Power BI Dashboard*

Business Implications

The insights generated from this project have direct implications for product and business strategy. A high session adoption rate demonstrates that users are receptive to new features, giving product managers confidence to continue rolling out updates and enhancements. At the same time, the paid share percentage highlights a strong but improvable area: with over 60% of users adopting paid features, targeted campaigns or personalized recommendations could help convert even more users into premium tiers.

The engaged users percentage indicates that adoption is not just surface-level, but reflects genuine and recurring use. This is a key signal for long-term retention strategies, as it shows that users are finding ongoing value in the platform. Meanwhile, the session trends by day and month provide a roadmap for understanding peaks and troughs in engagement. If sudden dips occur, teams can quickly investigate whether the cause was technical downtime, seasonal patterns, or shifts in user behavior.

By tying these insights back to the business, the dashboard can serve as both a monitoring tool and a decision-making guide. Executives and product teams alike can use this information to prioritize feature rollouts, adjust marketing strategies, and ensure system reliability, ultimately driving stronger user engagement and more sustainable growth.

Challenges Encountered

1. Large Event Parquet File

The most persistent hurdle came from the events.parquet dataset. Unlike the other parquet tables, this file was very large, schema-heavy, and difficult to query efficiently. In the beginning, every attempt to run even a simple query returned no rows. This was frustrating and time-consuming because it was not immediately clear whether the issue was the file itself, my code, or my understanding of Spark's streaming behavior. After trial and error, I discovered that the problem was partly due to how columns were being referenced and filtered. Renaming columns and carefully reviewing the schema helped me finally get usable output. This experience taught me how sensitive Spark can be to schema mismatches in large files, and it gave me practice in debugging systematically rather than guessing.

2. Limited Knowledge in Calculations and Streaming Concepts

Another challenge was my own limited knowledge of streaming semantics and Spark-specific calculations. Concepts like watermarking, windowing, and trigger once mode were new to me, and at first I did not fully grasp how they controlled the flow of data. For example, I initially tried running aggregations without a watermark, which caused my queries to return no results. I also struggled with when to use foreachBatch and how to safely write data so it wouldn't overwrite or corrupt existing files. Each failed run meant going back to the documentation, re-reading class materials, and often starting over. While frustrating, this forced me to slow down and understand not just what to type, but why those lines of code were necessary. It was a steep but valuable learning curve.

3. Visualization Connection Issues

Moving into visualization also presented difficulties. When I first connected Power BI to Databricks, I expected to immediately see my output tables. Instead, nothing appeared under the default Databricks database. This turned out to be a naming issue: the Delta tables I had created in my notebook weren't registered under the standard namespace. It took time to trace back where the outputs were being written in S3, confirm the Delta table names, and then reconfigure the connection in Power BI. While it added to the workload, this experience showed me how crucial naming conventions and data organization are in real projects. Without consistency, even working data can feel invisible.

4. File Size and Format Limitations

The final challenge was logistical but important. The instructor required submission of a Jupyter Notebook (.ipynb) file, but due to file size and platform constraints, I was only able to save my work in a .py script. Converting it to .ipynb was not feasible given the limitations I encountered, so I plan to note this in my submission. While this does not affect the logic of the project, it was a good reminder of how technical requirements (file type, format, platform) can sometimes be just as limiting as the actual coding challenges.

Lessons Learned

This project was not just about writing PySpark code; it was about developing a deeper understanding of how data pipelines function in practice. One of the most important lessons I learned was the significance of managing data schemas in a streaming context. When dealing with large files like the events.parquet dataset, even small inconsistencies in the schema can cause entire queries to fail or return empty results. Understanding how to define and preserve schemas became critical for ensuring that the pipeline could run consistently.

I also came to appreciate why Delta Lake plays such an essential role in modern data workflows. Delta provided the reliability needed for incremental writing, especially in a streaming scenario where data arrives continuously. Without Delta, it would have been far more difficult to manage issues like duplication, consistency, or the safe writing of micro-batches. Working through these concepts helped me see how Delta is not just a “nice-to-have” tool, but something that directly impacts the stability of the entire process.

Another major realization was that real-world pipelines require far more planning than I initially expected. Concepts such as handling empty batches, maintaining checkpoint locations, and using clear, consistent naming conventions all had a direct impact on the success of the pipeline. Early on, I overlooked these details, and as a result, some of my runs returned no output or caused unnecessary delays. Over time, I began to see that these seemingly small details are actually what make a production pipeline sustainable and dependable.

When it came time to connect the results to visualization tools like Power BI, I saw firsthand that dashboards are only as effective as the pipelines feeding them. Without clear table names, reliable Delta outputs, and properly written files, the dashboards were frustrating to build and interpret. This reinforced for me that data visualization is not a separate step at the end of the project; it depends directly on the structure and quality of the upstream pipeline.

Most importantly, I learned the value of patience and persistence in troubleshooting. Very few parts of this project worked perfectly on the first attempt. Whether it was streaming queries returning no rows, schema mismatches causing errors, or visualization tables failing to appear, progress often required repeated testing and iteration. Although this was frustrating at times, the process taught me how to approach problems systematically, break them into smaller parts, and refine my solution step by step. In many ways, the ability to troubleshoot and persist through these challenges became the most valuable outcome of the entire project.

Conclusion

This project brought together everything we covered in the course: cloud storage with AWS S3, large-scale processing in Databricks with PySpark, structured streaming with watermarks and windows, curated outputs in Delta Lake, and business-friendly dashboards in

Power BI. The pipeline worked end to end, turning raw session data into clear engagement metrics like adoption rates, paid share, and power-user activity. Along the way, I ran into real challenges with large parquet files, schema mismatches, and Power BI connections, but those obstacles became some of the most valuable learning moments. They forced me to slow down, understand Spark's behavior more deeply, and pay attention to practical details like naming conventions and safe writes.

Adding the machine learning demo brought the project full circle. Instead of stopping at descriptive KPIs, I was able to show how the same data could be used to make predictions about which customers are most likely to stay engaged. Even though the model was simple, it demonstrated the shift from looking backward at what already happened to looking ahead at what might happen. That step captured the real spirit of the assignment for me: not just building a pipeline that runs, but creating something that connects directly to business questions in a way that feels practical and forward-looking.

In the end, I'm proud of what I accomplished. I walked away with hands-on experience in designing, debugging, and delivering a big data pipeline that feels realistic to what data engineers and analysts tackle in the field. More importantly, I learned how patience, persistence, and clarity in design make all the difference when working with large, messy, and constantly moving data. This project showed me how the pieces fit together, and it gave me confidence that I can carry these skills into future work where big data and real business decisions meet.